Python Blueprints

Game Blueprint

**Class 1: Game**

While loop: exit

Attributes:

1. actions (RSPT)
2. Points for each

R = 2

S = -1

T = 3

P = 0

Class Game:

Def\_\_init\_\_(self,points, r: points=2, s: points= -1, t: points=3, p: points=0)

Self.\_\_points = { (C,C) : (r,r), (D, D): (p,p), (C,D): (s,t), (D,C): (t,s)}

def score(self, pair: Tuple[Action, Action]) -> Tuple[Score, Score]:

"""Returns the appropriate score for a decision pair.

Parameters

----------

pair: tuple(Action, Action)

A pair actions for two players, for example (C, C).

Returns

-------

tuple of int or float

Scores for two player resulting from their actions.

"""

return self.scores[pair]

Class 2: Strategy

Def \_\_init\_\_(self, other = false, strategy):

Self.strategy = C, D

Self.other = other

Def strategy(self):

def D(self):

print(score)

self.history.append()

def C(self):

print(score)

self.history.append()

Comp = strategy()

Class 3: Player

Attributes: History, Score,

Methods:

Def \_\_init\_\_(self, historySelf, historyOther, score, compStrategy, otherStrategy):

Self.score = score

Self.historySelf = [ ]

Self.historyOther = [ }

Self.compStrategy = [ ]

Self.otherStrategy = otherStrategy

# Combine Actions and Score into dictionary

Def score(self):

Def historySelf(self):

Def historyOther(self):

Def compStrategy(self):

class Copycat(Player):

"Player mimics move"

name = "Copycat"

def strategy(self,other):

#first move

if len(self.history) == 0:

return C

elif other.history[-1]== D:

return D

else:

return C

class Cooperator(Player):

"Player cooperates"

name = "Cooperator"

def strategy(self, opponent):

if len(self.history) == 0:

return C

else:

return C

class Cheater(Player):

"Player cheats"

name = "Cheater"

def strategy(self,opponent):

if len(self.history) == 0:

return D

else:

return D

class Grudger(Player):

"Player cooperates until opponent cheats"

name = "Grudger"

def strategy(self,opponent):

if len(self.history) == 0:

return D

if opponent.history[-1] = C:

return C

else:

return D

class Reformer(Player):

"Player cheats until opponent cooperates"

name = "Reformer"

def strategy(self,opponent):

if len(self.history) == 0:

return C

if opponent.history[-1] = D:

return D

else:

return C

Def otherStrategy(self):

Input = input(“C or D”)

While input ! C or D:

Ask again.

If c:

Append.Otherhistory[]

Print(Score)

If d:

Append.Otherhistory[]

Class 4: Match

Class Match:

Def\_\_init\_\_(

Listofcompstrat = [cooperator, copycat, cheater, grudger, reformer]

Game = [random.sample(listofcompstrat), otherstrategy)]